A Clustering Structure for Reliable Multicasting

Lenka Motyčková
ITN, Linköping University
Sweden

Esther Jennings[1]
Department of Computer Science
California State Polytechnic University, Pomona

Abstract

In reliable multicast, the multicast packets must be acknowledged. We propose a clustering structure which can be used by most of the existing reliable multicast protocols for collecting acknowledgments and for making local retransmissions.

Given a network $\mathcal{N}$ and a multicast routing tree (or a set of trees) T, we consider a subgraph G of $\mathcal{N}$ induced by the members of a multicast group. We then form disjoint clusters (local groups) of multicast receivers such that the receivers within a cluster are densely connected in G. The goal is to obtain a balanced clustering structure (dependent on the topology of G) such that the number of clusters is constant and the cluster size is kept low. This structure enables different clusters to process acknowledgments concurrently. It is also used to localize retransmissions. That is, when a packet is missed at a node, we will obtain the lost packet from another node which resides in the same cluster or at a nearby cluster whenever possible.

1 Introduction

Reliable multicast is needed in business, medical and military applications, where information needs to be disseminated reliably from multiple sources to all members of a multicast group. In source-initiated reliable multicast protocols, it is the source's responsibility to make sure that all the members of a multicast group have received the packets that it sent. On the other hand, in receiver-initiated reliable multicast protocols, it is the responsibility of each receiver to detect and to request any missed packets. A packet's loss can be detected by receivers if the packets are numbered serially. A positive acknowledgment (ACK) is used to inform the source that a packet is successfully received by the receiver(s). A negative acknowledgment (NAK) is used to inform the source of packet loss. If an ACK/NAK is sent directly to a source from every receiver, this might cause ACK-/NAK-implosion where the source spends nearly all of its time processing control messages. ACK-/NAK-implosion is an important issue in both sender-reliable and receiver-reliable protocols. For sender-initiated protocols, the source can release memory for the data which is already confirmed by the ACKs from all the receivers. A straightforward application of this principle, of course, causes ACK-implosion when the receiver group is large. Receiver-reliable protocols make the receivers responsible for the reliability of multicasting. That is, receivers will send NAKs for the missed packets only. However, this alone is not sufficient to insure reliability. Also, nodes which are incident to congested links might repeat NAKs and overload neighboring nodes.

Our proposed protocol avoids ACK-/NAK-implosion by using a clustered structure where ACK/NAKs are processed locally within each cluster whenever possible. Clustering is a known principle for achieving scalability. Our point is to define clusters so that they match a real topology of a network, i.e. they span subnetworks densely populated with multicast receivers. Constant network throughput is another goal of multicast protocols' design. By limiting the number of clusters and intra-cluster spanning tree degree, the load on the processors is kept independent of the number of receivers in the network.

1.1 Comparison to Previous Work

To avoid ACK-/NAK-implosion, most of the approaches proposed to date create local groups of receivers to obtain a tree structure where each local group contains a node and some of its neighbors. In the (rooted) shared ACK-tree of [10], a node forms a local group with B of its children, where B is a parameter. The clique clustering of [9] uses a greedy approach to create local groups of cliques. The structure we propose contains clusters that can be as dense as cliques or it can be sparse; the density of the clusters is controlled by a parameter which can be changed to suit different applications. That is, the structure we propose is more general than trees or cliques, and our structure offers more flexibility than previously pro-

[1]Partially supported by a grant from STINT, Sweden

posed structures. The main contribution of this paper is to propose a clustering structure which is suitable for gathering acknowledgments and for local retransmissions in reliable multicast. The features of this structure are: (1) the disjoint clusters enable local actions within each cluster (for sending ACKs and for retransmissions); this makes our protocol scalable, (2) optimal local routing (in terms of hop-count or delay) is provided within each cluster, (3) the intra-cluster links are used to propagate information to sources using a synchronization mechanism.

We note that the tree-based RMTP reliable multicast protocol also uses clustering [11]. However, their clusters have only depth one. Our cluster depth is controlled by a density parameter k. Also, the *designated routers* (cluster leaders) in RMTP are chosen statically for the multicast session. As an open problem, it is proposed that cluster leaders should be selected dynamically based on the actual network topology; our algorithm is based on the idea of selecting leaders in relatively densely populated areas. The RMTP protocol uses its tree topology as a logical structure which is not used for communicating acknowledgments. Acknowledgments are unicast between corresponding nodes, so the protocol relies heavily on unicast routing tables. In the case of concurrent sessions, special routing for retransmissions must be used in RMTP. The structure we propose can be used by any reliable protocol that handles concurrent multicast sessions without additional modifications. Our structure is general and one can apply different acknowledgment techniques to the structure. That is, any reliable multicast protocol can be imposed on this structure. For the sake of illustration, we propose a simple protocol for reliable multicast to show how this works on our structure. The proposed protocol has a congestion control mechanism and does not require unbounded memory as is required by some source-based tree protocols and by LBRM[8]. In addition, our structure supports concurrent acknowledgment gathering by using a heartbeat algorithm similar to synchronizers[2]. The heartbeat algorithm[13] delivers acknowledgment packets over the entire multicast group. Therefore, we do not need specific routings for each source during concurrent multicasting. This solves the known problem of managing concurrent sessions under the condition that the source does not know the identities of the receivers and the receivers do not know the location of each source.

1.2 Features of the Cluster Structure

The criteria suggesting a clustering topology are similar to the one proposed for synchronizers[2]. We use the heartbeat mechanism[2] and fast communication over the intra-cluster spanning trees to acknowledge packets. The structure we use is a *shared cluster graph* where the clusters are densely connected internally and inter-cluster links are sparse. This structure has the following desirable properties.

Good Scalability: The state information kept at each receiver is independent of the number of receivers. For intra-cluster routing, each node only keeps the information of its parent and children in the intra-cluster spanning tree. For inter-cluster routing, boundary nodes keep the information concerning edges leading to neighboring clusters.

Topology Preserving Clustering: The formation of local groups is not artificial but truthfully reflects the actual connectivity topology of the multicast group.

Optimal Routing within Clusters: For the intra-cluster trees, we consider two optimality criteria: hop-count and delay. To optimize hop-count, we compute a breadth-first-search spanning tree as the intra-cluster tree using the algorithm of [1]. To optimize delay, we use the Echo algorithm of [4].

Concurrent Sessions: The heartbeat algorithm supports concurrent sessions automatically. We do not need to compute routing tables with respect to each source nor do we need to compute specific labelings for routing.

2 Topology for Routing Acknowledgments

We organize the receivers of a multicast group into a shared ACK-cluster structure, built on top of the multicast routing tree(s) provided by best-effort multicast routing protocols such as DVMRP [5], PIM [6], CBT [3], OCBT [12]. We consider concurrent multicast sessions. These multicast sessions may use a shared multicast routing tree or several source-based multicast routing trees. The multicast tree(s) are used for data routing; our clustering structure is used to collect acknowledgments for the multicast packets.

Given an underlying shared multicast tree, or a set of source-based multicast trees, we define our graph G as the (connected) subgraph (of the network) induced by the vertices of the underlying multicast routing tree(s). Note that, G contains all the vertices of the multicast routing tree(s) plus all other network connections induced by these vertices. That is, an edge of G can be an IP-connection not used by any multicast routing tree(s). Since receivers are nodes that are directly connected to routers (as peripheral subtrees), we only consider router nodes in our graph G.

2.1 Clustering

The main idea of clustering is to identify the dense subgraphs and to make each into a cluster. If an area is densely populated with receivers, we can impose a condition for clustering such that the diameter of the cluster is logarithmic in terms of the number of cluster nodes. This type of clustering has been widely used; an example in the γ-synchronizer by Awerbuch [2]. By choosing an appropriate density parameter k (which may vary for different clusters), we can obtain a cluster graph with low cluster diameter (hence short delays for gathering acknowledgments and for retransmissions), and sparse inter-cluster connections (hence low communication cost for intercluster acknowledgments).

In our clustering method, we grow the clusters concurrently. First, we identify the sparse parts and the dense parts of the graph concurrently according to the prespecified density parameter k, ($1 < k \leq n$, where n is the multicast group size). Using this parameter, each node can determine whether it is in the dense or the sparse parts of the network. For each dense part (a cluster), we build an intra-cluster spanning tree.

The construction of the shared ACK-cluster structure is activated when a node receives the first multicast packet from a source. Then each member can start to compute the population around itself. The population around a node is computed by an expanding ring search. Each node counts the number of group members (population) at distance one from itself. If the population exceeds the density parameter k, then the node continues to count the number of group members at distance two from itself. If the population at distance two exceeds k^2, then the node continues to distance three. We continue in this way until we come to distance d where the population is less than k^d. The nodes which are surrounded by sufficient number of nodes within a certain radius (controlled by the parameter k) are called *cluster* nodes. The other nodes which are in the sparse parts of the graph are called *solitary* nodes. A solitary node is regarded as a cluster containing the single node. Solitary and cluster nodes are identified concurrently as soon as they receive the first multicast packet.

2.2 Intra-cluster spanning tree

Once the cluster and solitary nodes are identified, let G' be the subgraph of G induced by the cluster nodes. Then, each connected component of G' is a cluster. For each cluster, we construct a intra-cluster spanning tree as follows. Each boundary node of a cluster starts to grow its spanning tree using the Echo algorithm of [4]. The Echo algorithm has its own acknowledgment mechanism (the echo) which does not use unicast routing tables. A *boundary* node is a node which has an edge whose other end-point belongs to another cluster. As several boundary nodes may start growing a spanning tree rooted at themselves, whenever two trees meet, one of them is defeated. The defeated tree stops growing and is abandoned. To decide which tree is the winner, we assume that each node has a unique identity number (e.g., its address) and we deterministically let the tree whose root has the smallest identity be the winner.

Now, if the delays on all the links are exactly the same, then the spanning tree built by the Echo algorithm is both a breadth-first-search (BFS) tree (minimum hop-count) and a shortest-path spanning tree (minimum delay). However, if the delays on the links are arbitrary but finite, then the Echo algorithm builds the tree as fast as possible, but it does not guarantee the tree to be a BFS tree. If the optimization criterion is to minimize the hop-count, then we would build a breadth-first-search spanning tree using the distributed algorithm in [1]. First, we use the Echo algorithm to select one border router as the winning node. Then, we build a breadth-first-search spanning tree rooted at that node.

To achieve good throughput, we need to bound the number of the clusters. For this purpose we may need to have a different density parameter k at different parts of the network according to their population with multicast group members. Assume for example that we select a value of k so big that the algorithm would create too many small clusters at sparsely populated parts of the network. To keep the number of clusters bounded by a constant we need to reduce the value of k. In this case another problem may occur: in the densely populated parts of the network we get big clusters that would be difficult to manage locally. The solution lays in "tuning" the parameter k so that it is not uniform for the whole network. We select a bigger value of k for dense subgraphs and a smaller k for sparse.

2.3 Cluster leader and logging-server

In each cluster, the cluster leader is chosen as the designated router (DR). The DRs are responsible for processing ACKs and performing retransmissions of missing packets in their local groups (clusters). Intra-cluster communication uses the intra-cluster spanning tree described in Subsection 2.2. For inter-cluster communication, if there are several links connecting a pair of clusters, then only one of these links is chosen as the

preferred link. Selection is made in designated routers, so that exactly one preferred link is chosen for each neighboring cluster. Inter-cluster communication uses preferred links only. DRs report their existence to active sources that might need to have a count of them when checking fast-ACKs/NAKs from clusters. This requirement might be relaxed depending on a specific protocol. For better reliability, each cluster leader has a backup node called a *logging-server* [8].

3 Reliable Multicasting

Levine, et al.[10], have shown that negative acknowledgment with periodic polling (NAPP-protocol) is superior to many other reliable multicast protocols proposed. Among the protocols reviewed in [10], the NAPP-protocol is the most scalable approach with respect to the number of receivers and provides the highest maximum throughput. We illustrate a usage of our structure for NAPP-protocol. Levine, et al. use a shared ACK-tree for routing acknowledgments. In this paper, we apply a clustering technique that scales better than a shared tree in large networks because we strive to obtain a balanced clustering of the multicast group. In the shared tree, ACK packets are aggregated from the leaves to the root of the tree. In comparison to the ACK-tree, our clusters are larger structures than the local groups (of depth one) in the shared ACK-tree because each of the clusters may have an internal spanning tree having more than one layer of nodes. Thus, using our clusters enables more parallelism in gathering ACKs than on a shared tree.

Our structure enables the clusters to gather ACKs concurrently. In a shared ACK-tree, ACKs have to be delivered to different sources, and the routing of these can be complicated. The routes might need to be relabeled whenever the network changes and therefore topology information must be broadcast over the network and routing tables must be updated. Our structure does not require any complicated routing scheme. In case of topology changes, only local updates are made in the cluster structure.

3.1 Sending Acknowledgments and Retransmission

Similar to other protocols (e.g. the protocol in [10]), we view a reliable multicast protocol as having two windows: a congestion window (cw) which advances based on receiver feedback concerning transmission pace and error detection, and a memory allocation window (mw) which advances based on receiver feedback concerning whether data can be erased from memory.

The basic strategy is to use two types of acknowledgments to advance these windows separately. The first type (fast acknowledgment, Fast-ACK/NAK) is sent immediately to advance the cw of the source and for requesting retransmission when necessary. Fast-ACK/NAK enables fast retransmissions and paces the rate of multicast packets from the source (congestion window). The second type (aggregated acknowledgment, aggregated-ACK) is used to confirm to a source that a packet is received by *all* the receivers of the multicast group so that the source can delete the packet (group of packets) from its memory. A cluster leader keeps packets in its memory until they are successfully delivered to all the members of its cluster. The logging-servers keep a duplicate of the information stored at their respective cluster leaders. Hence, the cluster leaders must inform the logging-servers about window advancements.

3.1.1 Fast-ACK/NAK

These acknowledgments are sent immediately after detection of a missing packet (NAK) or a successful receipt of a packet (fast-ACK). Within a cluster, the members send their fast-ACK/NAK to the cluster leader using the intra-cluster spanning tree. Each node sends its fast-ACK/NAK without waiting for the acknowledgment from its children. Fast-ACKs resp. fast-NAKs for the same packets received from the node's children are suppressed (not forwarded upward the intra-cluster tree) as long as they are of the same type, or a fast-NAK is received after a node has already sent off a fast-ACK. In the later case, a retransmission immediately occurs. In the case that a node sends a fast-NAK first and then gets a fast-ACK from any of its children, the fast-ACK is propagated over the intra-cluster tree and retransmissions occur. This implies that a cluster leader may receive up to two fast-ACK/NAK messages from each of its children and the second of them, the correcting fast-ACK, may be delayed. Therefore, a time constant T (linearly proportional to the delay on the diameter of the cluster) might be applied by a cluster leader when the leader is waiting for fast-ACKs from cluster members. After this time-out, the cluster leader unicasts the fast-NAK to a source. When sending a fast-ACK/NAK, a node includes the source identity, the packet identity and its own identity in the message. The cluster leader (DR) unicasts a fast ACK/NAK to the corresponding source. When a source receives the number of fast-ACKs exceeding the estimated lower bound of the number of clusters, then it advances its congestion window.

3.1.2 Aggregated ACK

The Aggregated-ACK is used to confirm reliable receipt of a packet by all the receivers of a multicast group; it confirms that there will be no more retransmission requests by NAKs. NAKs are not completely reliable because they can be lost. Therefore, we need to couple this with a time-out scheme. If a timer expired and no fast-ACK/NAK is received for a packet, then the packet is assumed to be lost and will be multicast again.

For aggregated-ACKs, we apply a hierarchical aggregation mechanism. First each cluster member must report aggregated-ACKs to its cluster leader. These aggregated-ACKs are collected on the intra-cluster tree from the leaves to the root. A node sends aggregated-ACKs after receiving it from all its children. Then cluster leaders synchronize with neighboring clusters to advance the memory window (release the storage of acknowledged packets and history information from its memory). The synchronization mechanism works as follows in two phases. In phase one, a leader sends aggregated-ACKs to all its neighboring clusters, and waits to receive aggregated-ACKs as replies. In the second phase (after receiving an aggregated-ACK as a reply in phase one from each neighboring cluster), a cluster leader sends final aggregated-ACKs to all neighboring clusters over preferred links. After receiving final aggregated-ACKs from all neighboring clusters, the cluster leader knows that it is safe to advance the memory window, and it notifies the local source.

3.1.3 Retransmission

Within the boundary of a cluster, a missed packet is retransmitted on the intra-cluster spanning tree from the cluster leader or between neighbors. Otherwise, the cluster leader requests the missing packet from neighboring clusters first. If this also fails, the cluster leader unicasts to the source to request retransmission. If the source did not receive the fast-ACK/NAK from all the cluster leaders (DRs) after a timeout, the packet is assumed to be lost and the source will retransmit the packet.

3.2 Concurrent Multicasting

Our cluster structure supports concurrent multicast sessions because we distribute aggregated-ACKs over the entire multicast group. That is, the aggregated-ACKs for packets are synchronized over all the clusters, so the active sources within each cluster are informed of the aggregated-ACKs. Therefore, we do not need to set up any specific routings to send aggregated-ACKs to each source. This is the major difference between our proposed protocol and other protocols.

3.3 Throughput Analysis

The tree-based [11] and tree-NAPP [10] protocols have constant throughput regardless of the number of receivers. The cluster structure is created using the parameter k to balance the number of clusters and the number of nodes in each of the clusters. In wide area networks, the degree of each node is bounded by K_{max}, where K_{max} is usually a constant. That is, the entire network is usually not a single large clique nor a network with very high connectivity.

By choosing the parameter k carefully in accordance to the graph topology, we achieve a good balance between the number of clusters and the cluster size while the number of clusters is bounded by a constant.

To analyze the throughput of reliable multicast protocols, we use the same model as [10]. The throughput of a protocol is a function of the number of packets that have to be processed at a given node in order to multicast a data packet. For this analysis, we assume that all loss events at any node in the multicast are independent and the probability of packet loss is p for any node. We avoid complicated analysis by assuming no loss of fast-ACK/NAK nor aggregated-ACKs. The following analysis is made with respect to the processing cost required at a node to successfully multicast a packet to all members.

Source: We send fast-NAKs through cluster leaders where each cluster leader represents a large number of receivers; this will dramatically decrease the number of fast-ACK/NAKs received. The number of cluster leaders is bounded by a constant. Then the source retransmits lost packets if it receives NAKs which cannot be locally handled in the clusters. The number of NAKs received at a source is dependent on the loss probability and the number of cluster leaders. A source node is also a cluster node, so it participates in the gathering of aggregated-ACKs. This is carried out on the intra-cluster spanning tree. Assumed that the degree of a spanning tree is bounded by the constant K_{max}, then it receives a constant number of ACKs from its children and sends one aggregated-ACK to its parent. Thus, the load at a source is bounded by the number of clusters which is claimed to be constant.

Cluster leader: A leader is also a cluster node so it participates in the gathering of aggregated-ACKs. Assumed that a degree of a spanning tree is bounded by the constant K_{max}, then it receives a constant

number of ACKs from its children and sends one aggregated-ACKs to a constant number of children (towards the border nodes which are incident to preferred links and to the active sources). A cluster leader performs local retransmissions, and it may fetch packets which are fast-ACKed from cluster nodes. The number of retransmissions is bounded by the degree of intra-cluster tree, K_{max}. By choosing K_{max} as a constant, the load of the cluster leader is bounded by constant.

Receiver or Hop-node: A cluster node participates in the gathering of aggregated-ACKs. Assumed that a degree of a spanning tree is bounded by the constant K_{max}, then it receives a constant number of ACKs from its children and sends one aggregated-ACK to its parent. So, the load of a receiver or a hop-node is constant.

4 Conclusion

In this paper, we have proposed a clustering structure which is suitable for collecting acknowledgments and for local retransmission of multicast packets. This structure can be used by most of existing reliable multicast protocols. We have proposed a simple multicast protocol which uses this shared-cluster structure and have shown that the protocol has good scalability. It also avoids ACK-/NAK-implosion by carefully choosing the density and cluster size parameters.

We also analyze its throughput and found that nodes in the cluster structure have a constant throughput. This structure is scalable because we distribute the responsibility of gathering acknowledgments and retransmissions to the clusters. This allows more parallelism in acknowledgment packets gathering and localize the retransmissions. Also, using this structure saves memory because we do not need to compute/update routing tables with respect to each source during multiple concurrent sessions.

Adding a new receiver into a cluster does not change a diameter of the cluster. A receiver leaving a multicast group may cause splitting a cluster into two; in this case a local reconfiguration of the topology is necessary.

We balance the number of clusters and their size while keeping the number of clusters constant. The first issue is to distribute the load evenly over all involved routers. The second one is to prevent multicast sender from being overloaded by ACK/NAK packets implosion. These properties make a reliable multicast protocol running on the top of the proposed clustering structure, scalable.

References

[1] Awerbuch, B., Gallager, R.: "A new distributed algorithm to find breadth first search trees", IEEE Transactions on information theory, vol. IT-33, no. 3 (1987) 315–322.

[2] Awerbuch, B.: "Complexity of network synchronization", Journal of the ACM, vol. 32, No. 4, Oct. (1985) 804–823.

[3] Ballardie, T., Francis, P., Crowcroft, J.: "Core based trees (CBT): An architecture for scalable inter-domain multicast routing", Proc. ACM SIGCOMM (1993) 85–95.

[4] Chang, E. J. H.: "Echo algorithms: depth parallel operations on general graphs", IEEE Trans. on Software Engineering, vol. SE-8, no. 4, July (1982) 391–400.

[5] Deering, S., Cheriton, D.: "Multicast routing in datagram inter-networks and extended lans", ACM Trans. on Comp. Sys., vol. 8 (1990) 85–110.

[6] Deering, S., et al.: "An architecture for wide-area multicast routing", Proc. ACM SIGCOMM (1994) 126–135.

[7] Floyd, S., Jacobson, V., Liu, C.-G., McCanne, S., and Zhang, L.: "A reliable multicast framework for light-weight sessions and application level framing", Proc. ACM SIGCOMM (1995) 342–356.

[8] Holbrook, H., Singhal, S., Cheriton, D.: "Log-based receiver-reliable multicast for distributed interactive simulation", ACM SIGCOMM (1995) 328–341

[9] Krishna, P., Vaidya, N., Chatterjee, M., Pradhan, D.: "A cluster-based approach for routing in dynamic networks", ACM SIGCOMM, Computer Communication Review, Apr. (1997).

[10] Levine, B., Lavo, D., Garcia-Luna-Aceves, J.J.: "The case for reliable concurrent multicasting using shared act trees", Proc. ACM Multimedia, Nov. (1996).

[11] Lin, J. C., Paul, S.: "RMTP: a reliable multicast transport protocol", Proc. INFOCOMM (1996).

[12] Shields, C.: "Ordered core based trees", Master's thesis, University of California – Santa Cruz (1996)

[13] Tel, G.: "Introduction to distributed algorithms", Cambridge University Press (1994).